PROYECTO FINAL:
VISIÓN ARTIFICIAL Y RECONOCIMIENTO DE PATRONES

RECONOCIMIENTO DE OBJETOS

PEDRO LÓPEZ MARÍN

Contenido

Tiny-Yolo v3

1. INTRODUCCIÓN

El objetivo de este proyecto es crear un detector de objetos en imágenes, para tomar estas imágenes usaremos el módulo de cámara de la raspberry. Estas imágenes las pasaremos por nuestro algoritmo y nos indicará los objetos que hay en la foto, para señalar los objetos dibujará un cuadro delimitador a través de estos y les pondrá su etiqueta, es decir, que objeto es.

YOLO (YOU Only Look Once)

Explorando varias alternativas, la mejor opción para este caso es YOLO. Este algoritmo tiene la ventaja que solo necesita "ver la imagen una vez", lo que se traduce en mayor rapidez y menos coste computacional (mejor para la raspberry) aunque por contrapartida tiene que es menos exacto.

De forma rápida, el funcionamiento es el siguiente:

1. Divide la imagen en una cuadricula de SxS (imagen izquierda)
2. En cada cuadrícula predice N posibles cuadros delimitadores y calcula la puntuación de cada una de ellas (calcula SxSxN cajas diferentes). (imagen centro)
3. Se eliminan las cajas que estén por debajo de un umbral.
4. Cajas restantes: se aplica "non-max-suppression" para eliminar posibles objetos que fueron detectados por duplicado y dejar el más exacto (imagen derecha).

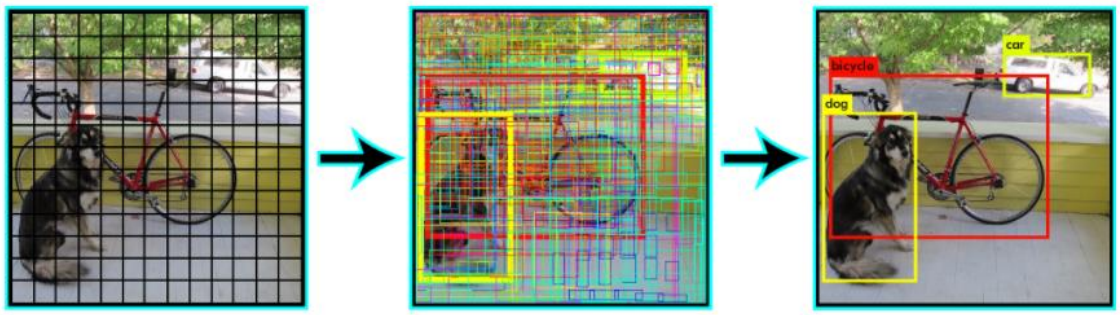


Ilustración 1. Algoritmo YOLO

VERSIONES

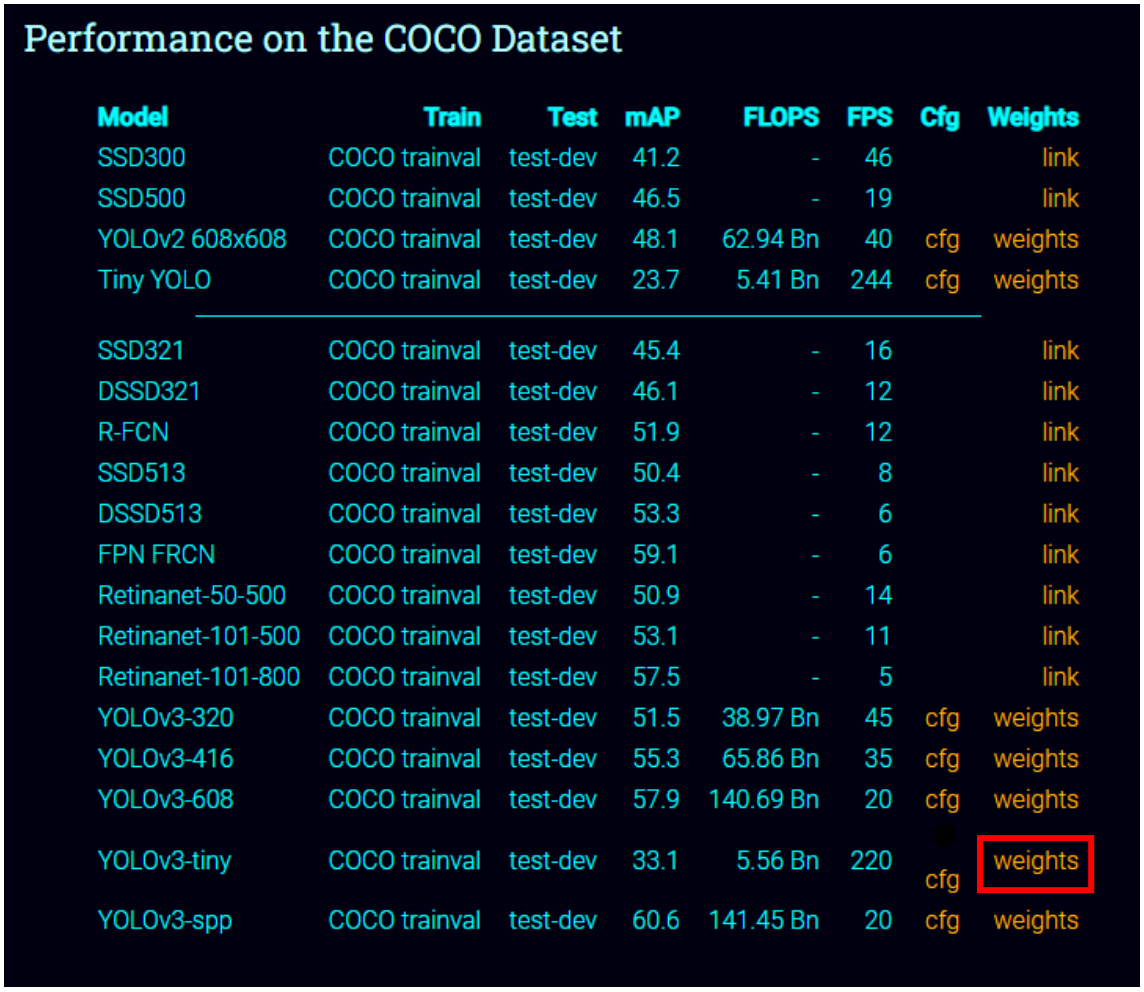
Hay diferentes versiones de YOLO desde v1 a v4, siendo esta última la más moderna y cada una con su respectiva versión reducida (tiny) que es la indicada para dispositivos con pocos recursos como es el caso de las raspberry.

Se pueden usar las versiones "completas" para la raspberry, pero el resultado sería mucho más lento y no es nada recomendable para video, para imágenes no habría problema, solo tarda más en procesar.

Para este proyecto se ha desarrollado la versión 3 (tiny). Aunque la 4 es la más moderna, apenas había documentación, por eso he optado por la 3. También se ha usado la versión versión "completa" para comparar resultados en imágenes.

OBTENCIÓN DE LOS PESOS

Para este proyecto se usan unos pesos de modelos ya preentrenados, se pueden encontrar en la web oficial de YOLO (1). Estos pesos son para el dataset COCO, por tanto, podremos diferenciar entre 80 tipos de objetos diferentes.



Performance on the COCO Dataset

Model	Train	Test	mAP	FLOPS	FPS	Cfg	Weights
SSD300	COCO trainval	test-dev	41.2	-	46		link
SSD500	COCO trainval	test-dev	46.5	-	19		link
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40	cfg	weights
Tiny YOLO	COCO trainval	test-dev	23.7	5.41 Bn	244	cfg	weights
SSD321	COCO trainval	test-dev	45.4	-	16		link
DSSD321	COCO trainval	test-dev	46.1	-	12		link
R-FCN	COCO trainval	test-dev	51.9	-	12		link
SSD513	COCO trainval	test-dev	50.4	-	8		link
DSSD513	COCO trainval	test-dev	53.3	-	6		link
FPN FRCN	COCO trainval	test-dev	59.1	-	6		link
Retinanet-50-500	COCO trainval	test-dev	50.9	-	14		link
Retinanet-101-500	COCO trainval	test-dev	53.1	-	11		link
Retinanet-101-800	COCO trainval	test-dev	57.5	-	5		link
YOLOv3-320	COCO trainval	test-dev	51.5	38.97 Bn	45	cfg	weights
YOLOv3-416	COCO trainval	test-dev	55.3	65.86 Bn	35	cfg	weights
YOLOv3-608	COCO trainval	test-dev	57.9	140.69 Bn	20	cfg	weights
YOLOv3-tiny	COCO trainval	test-dev	33.1	5.56 Bn	220	cfg	weights
YOLOv3-spp	COCO trainval	test-dev	60.6	141.45 Bn	20	cfg	weights

Ilustración 2. Página de los pesos de YOLO, podemos ver el rendimiento de los diferentes modelos

Debemos descargar el archivo que pone weights, que son los pesos para nuestro modelo, en este caso el YOLOv3-tiny.

NOTA:
Vemos otro archivo .cfg, que es para la configuración del modelo, está preparado para descargar el modelo del repositorio del autor y ya seleccionar la configuración y los pesos correspondientes. Este repositorio está escrito en C, por tanto, nos olvidaremos de él y se hará una implementación en Python.

2. ENTORNO DE TRABAJO

Vamos a trabajar con Google Colaboratory. Lo primero es crearnos en nuestro Drive un directorio de trabajo para tenerlo todo más organizado. Colocaremos el archivo de los pesos en este directorio.

En la Raspberry

Creamos un directorio en el que guardaremos el modelo creado en formato .h5, el código del programa y las imágenes.

```
# Salvar el modelo como .h5
model.save('{}'.format('model_yolo.h5'))
```

Ilustración 3. Guardar el modelo

3. CÓDIGO - YOLO

FUNCIÓN SET_WEIGHTS

Esta función sirve para extraer los pesos del archivo .weights y establecerlos en nuestro modelo (2). A esta función se le llamará en cada capa.

MODELO TINY_YOLOv3

Crearemos el modelo o red convolucional con keras, es importante respetar el número de capas y tipo de estas para poder establecer los pesos. Debemos seguir el esquema de la Ilustración 4. Capas TinyYolov3A continuación, se detallan todos los tipos de capas (3):

Layer	Type	Filters	Size/Stride	Input	Output
0	Convolutional	16	3 × 3/1	416 × 416 × 3	416 × 416 × 16
1	Maxpool		2 × 2/2	416 × 416 × 16	208 × 208 × 16
2	Convolutional	32	3 × 3/1	208 × 208 × 16	208 × 208 × 32
3	Maxpool		2 × 2/2	208 × 208 × 32	104 × 104 × 32
4	Convolutional	64	3 × 3/1	104 × 104 × 32	104 × 104 × 64
5	Maxpool		2 × 2/2	104 × 104 × 64	52 × 52 × 64
6	Convolutional	128	3 × 3/1	52 × 52 × 64	52 × 52 × 128
7	Maxpool		2 × 2/2	52 × 52 × 128	26 × 26 × 128
8	Convolutional	256	3 × 3/1	26 × 26 × 128	26 × 26 × 256
9	Maxpool		2 × 2/2	26 × 26 × 256	13 × 13 × 256
10	Convolutional	512	3 × 3/1	13 × 13 × 256	13 × 13 × 512
11	Maxpool		2 × 2/1	13 × 13 × 512	13 × 13 × 512
12	Convolutional	1024	3 × 3/1	13 × 13 × 512	13 × 13 × 1024
13	Convolutional	256	1 × 1/1	13 × 13 × 1024	13 × 13 × 256
14	Convolutional	512	3 × 3/1	13 × 13 × 256	13 × 13 × 512
15	Convolutional	255	1 × 1/1	13 × 13 × 512	13 × 13 × 255
16	YOLO				
17	**Route 13**				
18	Convolutional	128	1 × 1/1	13 × 13 × 256	13 × 13 × 128
19	Up-sampling		2 × 2/1	13 × 13 × 128	26 × 26 × 128
20	**Route 19 8**				
21	Convolutional	256	3 × 3/1	13 × 13 × 384	13 × 13 × 256
22	Convolutional	255	1 × 1/1	13 × 13 × 256	13 × 13 × 256
23	YOLO				

Ilustración 4. Capas TinyYolov3

CONVOLUCIONAL

Esta capa es la que aplica los filtros. Cada capa convolucional debe ir seguida de una de bacth normalization (normalización en lote) y de la función de activación Leaky-ReLU.

```
# Layer 0
x = Conv2D(16, (3,3), strides=(1,1), kernel_regularizer=l2(regulador), weights=conv_weights, padding='same', name='conv_1', use_bias=False)(Input)
x = BatchNormalization(weights=bn_weight_list, name='norm_1')(x)
x = LeakyReLU(alpha=0.1)(x)
```

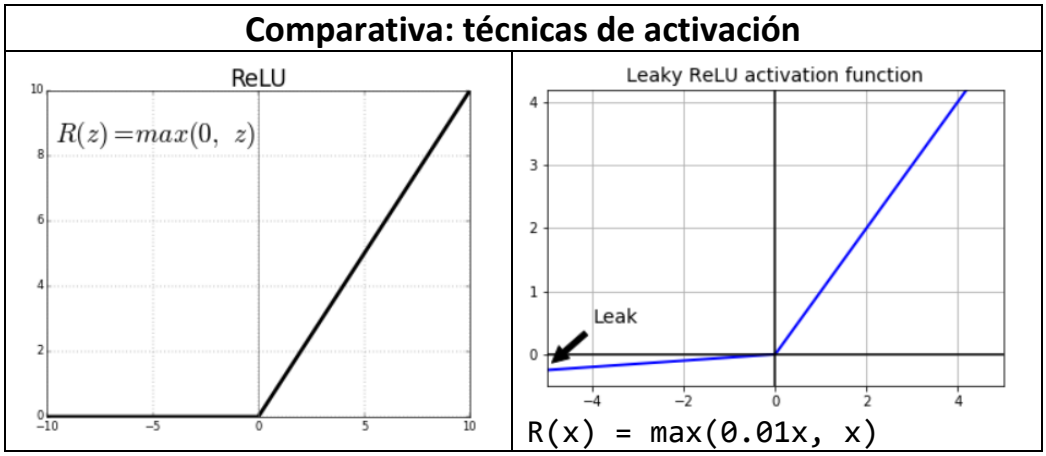
Ilustración 5. Ejemplo Capa Convolucional.

Capa Convolucional. En esta capa podemos distinguir que se usan 16 filtros (kernel) de 3x3, por lo tanto, la salida ahora será de 416 x 416 x 16. Usamos un stride de 1, es decir, el filtro se desplaza de píxel en píxel. El kernel_regularizer es para evitar el sobreajuste, usamos 0.01 que es el más común. Padding, es un método para añadir píxeles alrededor de la imagen para poder aplicar el kernel a los píxeles de los extremos, con same conseguimos que las salidas de la capa tengan las mismas dimensiones que sus entradas. También se llama la función de los pesos.

Vemos la entrada de esta capa que es Input: Es un tensor con el tamaño de entrada indicado (416 x 416 x3). Esto solo en el caso de la primera capa, en el resto la entrada será la salida de la anterior.

Bath Normalization. Esta capa siempre se suele poner entre las neuronas (salida convolucional) y la función de activación (Leaky ReLU, se detalla después) con el objetivo de normalizar las activaciones de salida para facilitar la labor de aprendizaje del modelo. A esta capa también hay que asignarle los pesos correspondientes.

Capa de Activación. Para este caso se emplea la función de activación Leaky RELU. Esta es la más usada junto a ReLU, pero con la ventaja de que LeakyReLU también aprende (aunque menos) cuando el valor es negativo, es decir, modifica el peso (4). El valor que le daremos al Alpha, el coeficiente de pendiente negativo es 0.1.



MAXPOOL

Son capas de "reducción" y nos ayudan a acelerar el proceso de cómputo para procesar los datos.

```
# Layer 1
x = MaxPooling2D(pool_size=(2, 2), name='Max_2')(x)
```

Ilustración 6. Ejemplo Capa MaxPooling

Básicamente esta capa aplica la técnica de Max-pooling con un tamaño de 2x2. Esto es que recorre las imágenes de entrada de 416 x 416, pero las recorre tomando 4 píxeles (2 de ancho y 2 de alto) y como en este caso es MAX. se queda con el mayor valor de esos 4. Con este proceso la imagen de entrada se reduce a la mitad, por lo que el tamaño de salida es de 208 x 208. Con lo cual podemos reducir el número de neuronas sin perder información, teóricamente.

UP-SAMPLING

Esta capa hace lo contrario que la anterior y lo que hace es duplicar filas y columnas, por eso, en esta capa tenemos una entrada de 13 x 13 y obtenemos una salida de 26 x 26. **Ver ilustración 2*

YOLO

Este modelo tiene dos capas de salida: yolo0 y yolo1. A efectos prácticos ambas son arrays de numpy de tamaño (13, 13, 3, 85) y (26, 26, 3, 85) respectivamente. Donde los 13 y 26 vienen del tamaño de la salida de la capa anterior (ancho y alto en píxeles); el 3 viene de RGB y el 85 viene del número de clases, para MSCOCO que son 80; + 4, de las coordenadas; + 1, de la puntuación de clase.

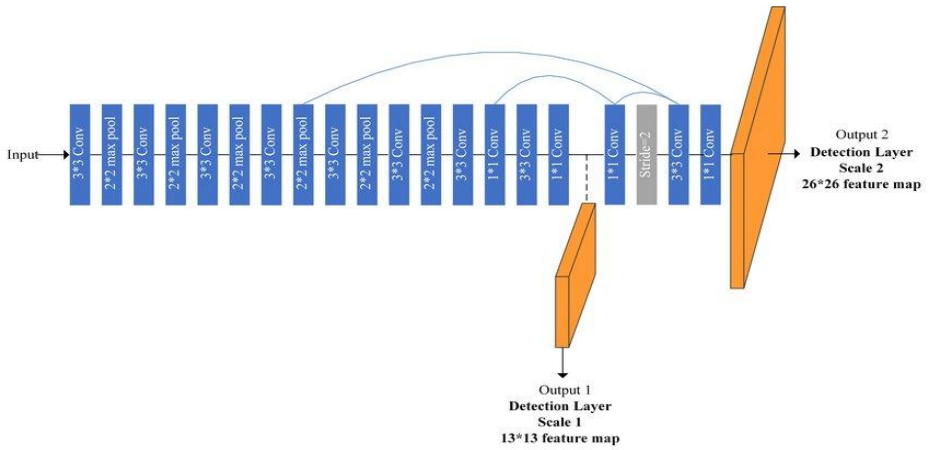


Ilustración 7. Arquitectura YOLO

ROUTE

Esta capa es como concatenar dos. Por ejemplo, para la capa 20 (Route 19 8), tomamos como entrada el resultado de las capas con índice 19 (corresponde con x, la salida de la capa anterior) y 8 (corresponde con route20, que es el nombre que se le ha dado) y las fusionamos en una sola capa con concatenate (5).

```
x = concatenate([route20, x])
```

Ilustración 8. Capa 20. Route 19 y 8

4. CÓDIGO – PROGRAMA RASPBERRY

FUNCIONES AUXILIARES

Estas son las funciones necesarias posteriores a la predicción para poder dibujar el cuadro delimitador y etiquetar el objeto (6).

BoundBox

Es una clase, define las esquinas de cada cuadro delimitador en el contexto de la forma de la imagen de entrada y las puntuaciones de clase. Las puntuaciones de clase podrían verse como la posibilidad de que un objeto detectado sea ese tipo.

- **get_label**: asigna la etiqueta.
- **get_score**: asigna el porcentaje de confianza

_sigmoid

Para acotar resultados entre 0 y 1. Se emplea para la predicción de coordenadas centrales del cuadro delimitador en la función `decode_netout()`.

Decode_netout

Decodificar la salida de la red. Recorre cada una de las matrices NumPy y va decodificando los cuadros delimitadores candidatos y las predicciones de clase en función del umbral indicado.

Para estas matrices, los 4 primeros elementos son las coordenadas del cuadro delimitador seguido de la puntuación y de las probabilidades de clase.

Correct_yolo_boxes

Esta función se encarga de hacer que los cuadros delimitadores predichos coincidan con el tamaño de la imagen original, recordemos que hay que adaptar la imagen a un tamaño de 416x416.

_interval_overlap

Comprueba si dos intervalos se superponen y nos devuelve la intersección. Esta función se usará para obtener la intersección de dos cuadros delimitadores*.

Bbox_iou

Intersección sobre la unión (IoU) es una métrica para medir la precisión de la localización y estimar los errores de localización en los modelos de detección de objetos. Nos da una estimación de que tan cerca está el cuadro delimitador de la predicción original.

IoU es el cociente entre la intersección de los cuadros delimitadores para una predicción particular y los cuadros delimitadores reales de la misma área, entre el área total cubierta por ambos cuadros, lo que se conoce como Unión.

$$IOU = \frac{\text{area of overlap}}{\text{area of union}} =$$

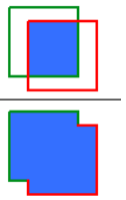


Ilustración 9. IoU explicación

Supress_nms (nom maximal boxes)

Toma la lista de cuadros delimitadores y un umbral. Lo que hacemos es, en vez de borrar las casillas superpuestas, le eliminamos la probabilidad para la clase superpuesta. Con esto conseguimos que las cajas sigan estando y se puedan usar para detectar otro objeto.

Load_Image

Esta función nos carga la imagen que deseamos y también se encarga de prepararla para poder trabajar con ella: la ajusta al tamaño 416x416, la convierte a un array de numpy y la normaliza.

Get_boxes

Esta función obtiene los resultados: caja, etiqueta y puntuación. Recorre todas las cajas que se le pasan por parámetro y comprueba si alguna etiqueta supera la puntuación del umbral, cuando la

encuentra devuelve esa caja, la etiqueta correspondiente y la puntuación. ¡OJO!, puede haber muchas etiquetas para una misma caja.

Draw_boxes

Traza los cuadros delimitadores con sus etiquetas sobre los objetos detectarlos en la imagen original. Hacemos uso de la librería Matplolib tanto para cargar la imagen y mostrarla, como para dibujar el cuadro (función Rectangle) y para poner la etiqueta.

CON LA PICAMERA

Para procesar imágenes que tomemos con nuestra raspberry, primero haremos una visión previa de 3seg para poder "apuntar". Después la guardaremos en la ruta de nuestro directorio de trabajo, debemos usar esta ruta también para luego cargar la imagen y procesarla.

```
# PiCamera
with picamera.PiCamera() as camera:
    camera.start_preview()
    time.sleep(3)
    camera.capture('/home/pi/Desktop/vision/images/imagen_PiCam2.jpg')
    camera.stop_preview()
```

Ilustración 10. Código PiCamera

ANTES DE LA PREDICCIÓN

Umbrales

Para el umbral de detección de objetos, debido al bajo poder computacional de la raspberry, las puntuaciones de los objetos serán bajas por tanto es recomendable usar un umbral bajo, haciendo pruebas recomiendo como mucho 0.3. Si lo ponemos más alto, se quedarán objetos sin detectar, pero si lo ponemos muy bajo acabará superponiendo cuadros delimitadores con otras etiquetas al mismo objeto.

Para el umbral de superposición de cuadros delimitadores es preferible dejarlo alto, sobre 0.6

Anclas

[[116,90, 156,198, 373,326], [30,61, 62,45, 59,119], [10,13, 16,30, 33,23]]

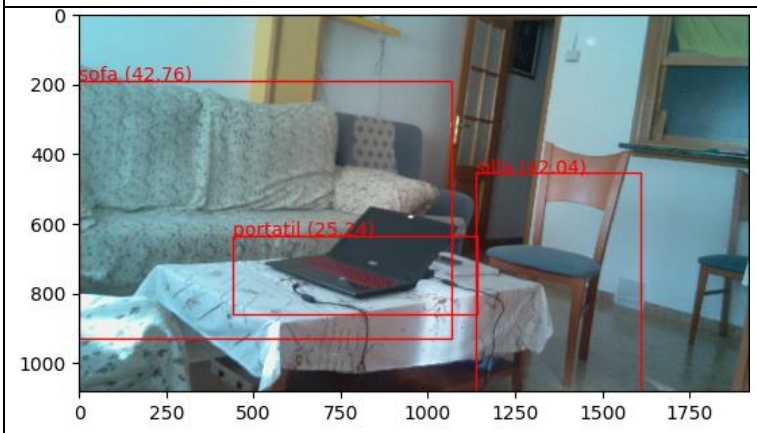
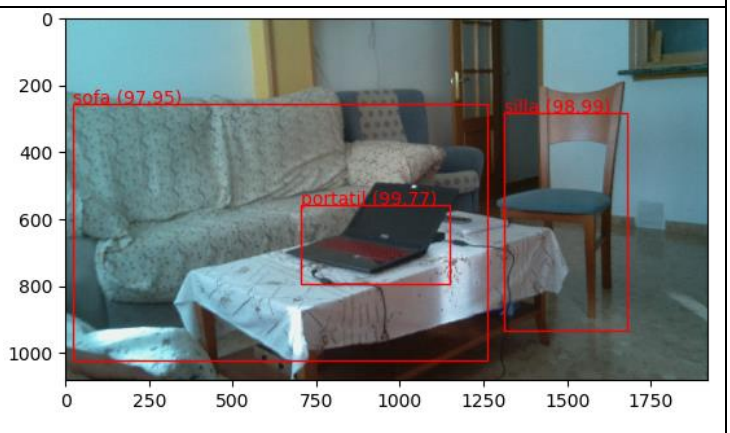
Etiquetas

Estas son las 80 etiquetas correspondientes al dataseet COCO:

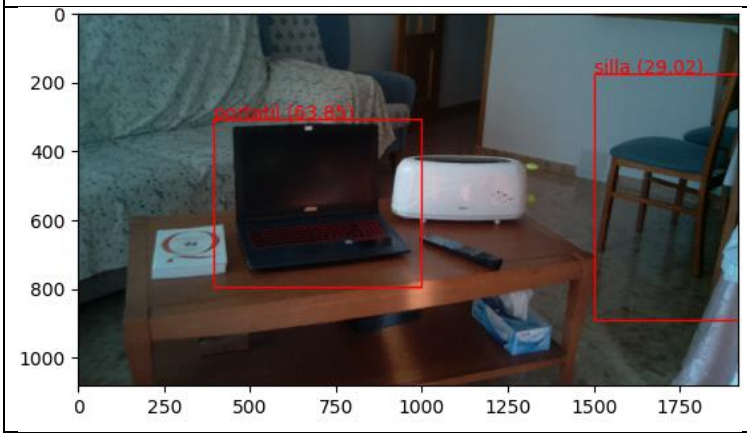
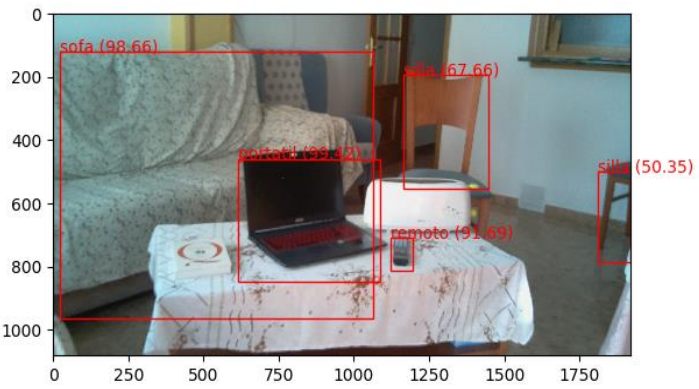
"persona", "bicicleta", "coche", "moto", "avión", "bus", "tren", "camión", "barco", "semáforo", "boca de incendio", "señal de stop", "parquimetro", "banco", "pajaro", "gato", "perro", "caballo", "oveja", "vaca", "elefante", "oso", "cebra", "jirafa", "mochila", "paraguas", "bolso", "corbata", "maleta", "frisbee", "esquis", "snowboard", "deporte de balon", "cometa", "bate besibol", "guante beisbol", "skateboard", "tabla de surf", "raqueta", "botella", "copa", "taza", "tenedor", "cuchillo", "cuchara", "bowl", "banana", "manzana", "sandwich", "naranja", "broccoli", "zanahoria", "hot-dog", "pizza", "donut", "tarta", "silla", "sofa", "maceta", "cama", "mesa", "inodoro", "tvmonitor", "portatil", "raton", "remoto", "teclado", "movil", "microondas", "horno", "tostadora", "lavabo", "frigorifico", "libro", "reloj", "jarron", "tijeras", "oso peluche", "secador", "cepillo dental"

5. RESULTADOS

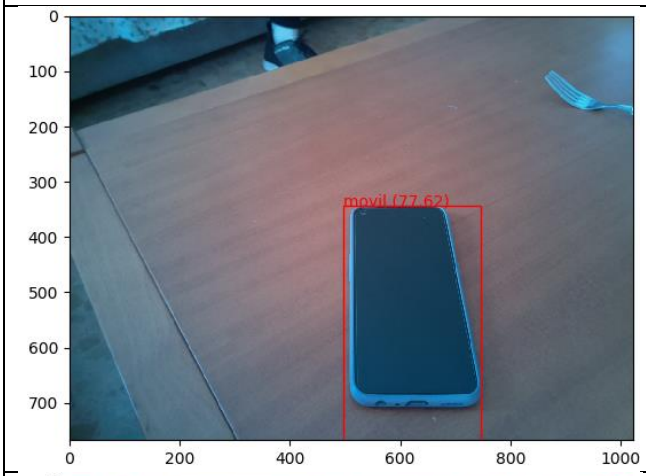
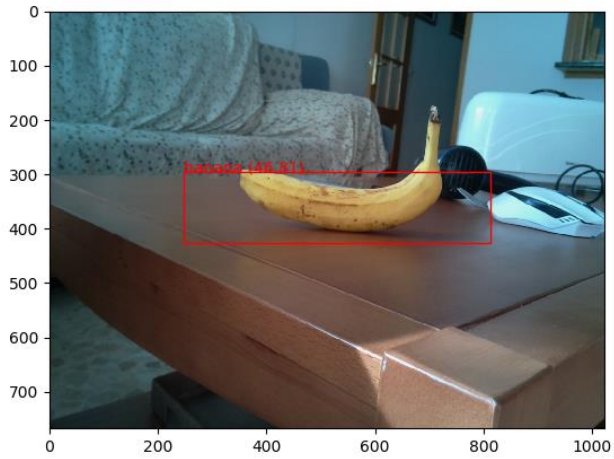
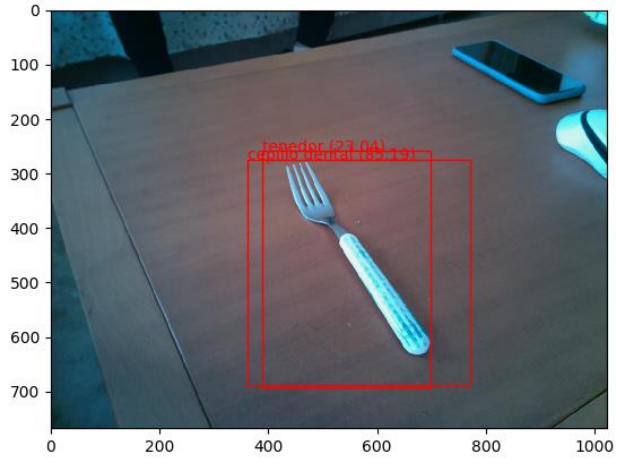
En esta primera parte se muestra una comparativa de la versión tiny frente al modelo “completo”

TINY-YOLO	YOLO
	

Como se puede observar ambas versiones detectan los mismos objetos. Se puede apreciar también que las cajas delimitadoras de YOLO son más precisas, también las puntuaciones de clase de los objetos son mayores, todas por encima del 95%, en contraste con las de tiny-yolo, que la más alta no llega ni al 45%.

TINY-YOLO	YOLO
	

Se han añadido algunos objetos más. En yolo es capaz de detectar el mando y el sofá (quizá en tiny no porque está muy cerca). Se ha probado a quitar el mantel de la mesa para que aumentase el contraste con el libro y la tostadora, pero no funcionó. Volvemos a ver que la versión YOLO es más potente, da más puntuación y además es capaz de detectar más objetos y objetos que se ven menos como la silla de la derecha.

<table>
<tr><th colspan="2">Con objetos individuales</th></tr>
<tr><td> 
</td><td>  </td></tr>
<tr><td> 
</td><td>Con objetos individuales parece que funciona mejor, incluso consigue mayores puntuaciones. Salvo el caso del tenedor en el que detecta el tenedor, pero también detecta un cepillo dental y con más puntuación que el tenedor.</td></tr>
</table>

6. FUNCIONAMIENTO

El programa es mainYOLO.py, cuando lo ejecutemos nos aparecerá lo que ve la cámara, tenemos 3segundos para apuntar lo que queremos y entonces capturará la imagen (tiene más ángulo que el que se ve en la previsualización). Después es todo automático, el programa ya se encarga de cargar y procesar la foto, cuando acabe nos aparecerá la imagen con los objetos detectados.

Nota: debemos tener el archivo model_yolo.h5 en el mismo directorio.

Bibliografía

1. Redmon, Joshep. pjreddie. [En línea] https://pjreddie.com/darknet/yolo/.

2. **Dennis.** GitHub. [En línea] https://github.com/xiaochus/YOLOv3/blob/db63e48f501a9019eb420f77dfc7fa6f44329270/yad2k.py.

3. Guzman, Edgar Roberto Silva. [En línea] https://biorobotics.fi-p.unam.mx/wp-content/uploads/Courses/reconocimiento_de_patrones/tutoriales/YOLO-Introducci%C3%B3n-e-implementaci%C3%B3n-.pdf.

4. Quora. [En línea] https://www.quora.com/What-are-the-advantages-of-using-Leaky-Rectified-Linear-Units-Leaky-ReLU-over-normal-ReLU-in-deep-learning.

5. [En línea] https://github.com/AlexeyAB/darknet/issues/279#issuecomment-397248821**.**

6. khandelwal, Renu. towards data science. [En línea] https://towardsdatascience.com/object-detection-using-yolov3-using-keras-80bf35e61ce1.